**Hibernate API**

Hibernate is an **O**bject-**R**elational **M**apping (ORM) solution for JAVA. It is an open source persistent framework created by Gavin King in 2001. It is a powerful, high performance Object-Relational Persistence and Query service for any Java Application.

Hibernate maps Java classes to database tables and from Java data types to SQL data types and relieves the developer from 95% of common data persistence related programming tasks.

Hibernate sits between traditional Java objects and database server to handle all the works in persisting those objects based on the appropriate O/R mechanisms and patterns.



Hibernate Advantages

* Hibernate takes care of mapping Java classes to database tables using XML files and without writing any line of code.
* Provides simple APIs for storing and retrieving Java objects directly to and from the database.
* If there is change in the database or in any table, then you need to change the XML file properties only.
* Abstracts away the unfamiliar SQL types and provides a way to work around familiar Java Objects.
* Hibernate does not require an application server to operate.
* Manipulates Complex associations of objects of your database.
* Minimizes database access with smart fetching strategies.
* Provides simple querying of data.

Supported Databases

Hibernate supports almost all the major RDBMS. Following is a list of few of the database engines supported by Hibernate −

* HSQL Database Engine
* DB2/NT
* MySQL
* PostgreSQL
* FrontBase
* Oracle
* Microsoft SQL Server Database
* Sybase SQL Server
* Informix Dynamic Server

Supported Technologies

Hibernate supports a variety of other technologies, including −

* XDoclet Spring
* J2EE
* Eclipse plug-ins
* Maven

Hibernate has a layered architecture which helps the user to operate without having to know the underlying APIs. Hibernate makes use of the database and configuration data to provide persistence services (and persistent objects) to the application.

Following is a very high level view of the Hibernate Application Architecture.



Following is a detailed view of the Hibernate Application Architecture with its important core classes.



Hibernate uses various existing Java APIs, like JDBC, Java Transaction API(JTA), and Java Naming and Directory Interface (JNDI). JDBC provides a rudimentary level of abstraction of functionality common to relational databases, allowing almost any database with a JDBC driver to be supported by Hibernate. JNDI and JTA allow Hibernate to be integrated with J2EE application servers.

Following section gives brief description of each of the class objects involved in Hibernate Application Architecture.

Configuration Object

The Configuration object is the first Hibernate object you create in any Hibernate application. It is usually created only once during application initialization. It represents a configuration or properties file required by the Hibernate.

The Configuration object provides two keys components −

* **Database Connection** − This is handled through one or more configuration files supported by Hibernate. These files are **hibernate.properties** and **hibernate.cfg.xml**.
* **Class Mapping Setup** − This component creates the connection between the Java classes and database tables.

SessionFactory Object

Configuration object is used to create a SessionFactory object which in turn configures Hibernate for the application using the supplied configuration file and allows for a Session object to be instantiated. The SessionFactory is a thread safe object and used by all the threads of an application.

The SessionFactory is a heavyweight object; it is usually created during application start up and kept for later use. You would need one SessionFactory object per database using a separate configuration file. So, if you are using multiple databases, then you would have to create multiple SessionFactory objects.

Session Object

A Session is used to get a physical connection with a database. The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database. Persistent objects are saved and retrieved through a Session object.

The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed them as needed.

Transaction Object

A Transaction represents a unit of work with the database and most of the RDBMS supports transaction functionality. Transactions in Hibernate are handled by an underlying transaction manager and transaction (from JDBC or JTA).

This is an optional object and Hibernate applications may choose not to use this interface, instead managing transactions in their own application code.

Query Object

Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects. A Query instance is used to bind query parameters, limit the number of results returned by the query, and finally to execute the query.

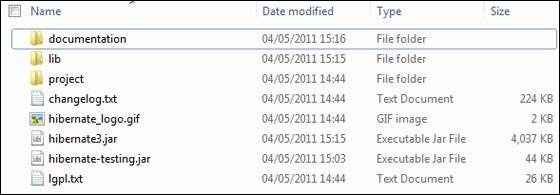
Criteria Object

Criteria objects are used to create and execute object oriented criteria queries to retrieve objects.

Downloading Hibernate

It is assumed that you already have the latest version of Java installed on your system. Following are the simple steps to download and install Hibernate on your system −

* Make a choice whether you want to install Hibernate on Windows, or Unix and then proceed to the next step to download .zip file for windows and .tz file for Unix.
* Download the latest version of Hibernate from <http://www.hibernate.org/downloads>.
* At the time of writing this tutorial, I downloaded **hibernate-distribution3.6.4.Final** and when you unzip the downloaded file, it will give you directory structure as shown in the following image



Installing Hibernate

Once you downloaded and unzipped the latest version of the Hibernate Installation file, you need to perform following two simple steps. Make sure you are setting your CLASSPATH variable properly otherwise you will face problem while compiling your application.

* Now, copy all the library files from **/lib** into your CLASSPATH, and change your classpath variable to include all the JARs −
* Finally, copy **hibernate3.jar** file into your CLASSPATH. This file lies in the root directory of the installation and is the primary JAR that Hibernate needs to do its work.

Hibernate Prerequisites

Following is the list of the packages/libraries required by Hibernate and you should install them before starting with Hibernate. To install these packages, you will have to copy library files from **/lib** into your CLASSPATH, and change your CLASSPATH variable accordingly.

|  |  |
| --- | --- |
| **Sr.No.** | **Packages/Libraries** |
| 1 | **dom4j**  XML parsing [**www.dom4j.org/**](http://www.dom4j.org/) |
| 2 | **Xalan**  XSLT Processor [**https://xml.apache.org/xalan-j/**](https://xml.apache.org/xalan-j/) |
| 3 | **Xerces**  The Xerces Java Parser [**https://xml.apache.org/xerces-j/**](https://xml.apache.org/xerces-j/) |
| 4 | **cglib**  Appropriate changes to Java classes at runtime [**http://cglib.sourceforge.net/**](http://cglib.sourceforge.net/) |
| 5 | **log4j**  Logging Faremwork [**https://logging.apache.org/log4j**](https://logging.apache.org/log4j) |
| 6 | **Commons**  Logging, Email etc. [**https://jakarta.apache.org/commons**](https://jakarta.apache.org/commons) |
| 7 | **SLF4J**  Logging Facade for Java [**https://www.slf4j.org**](https://www.slf4j.org/download.html) |

## Hibernate Properties

Following is the list of important properties, you will be required to configure for a databases in a standalone situation −

|  |  |
| --- | --- |
| **Sr.No.** | **Properties & Description** |
| 1 | **hibernate.dialect**  This property makes Hibernate generate the appropriate SQL for the chosen database. |
| 2 | **hibernate.connection.driver\_class**  The JDBC driver class. |
| 3 | **hibernate.connection.url**  The JDBC URL to the database instance. |
| 4 | **hibernate.connection.username**  The database username. |
| 5 | **hibernate.connection.password**  The database password. |
| 6 | **hibernate.connection.pool\_size**  Limits the number of connections waiting in the Hibernate database connection pool. |
| 7 | **hibernate.connection.autocommit**  Allows autocommit mode to be used for the JDBC connection. |

If you are using a database along with an application server and JNDI, then you would have to configure the following properties −

|  |  |
| --- | --- |
| **Sr.No.** | **Properties & Description** |
| 1 | **hibernate.connection.datasource**  The JNDI name defined in the application server context, which you are using for the application. |
| 2 | **hibernate.jndi.class**  The InitialContext class for JNDI. |
| 3 | **hibernate.jndi.<JNDIpropertyname>**  Passes any JNDI property you like to the JNDI *InitialContext*. |
| 4 | **hibernate.jndi.url**  Provides the URL for JNDI. |
| 5 | **hibernate.connection.username**  The database username. |
| 6 | **hibernate.connection.password**  The database password. |

## Session Interface Methods

There are number of methods provided by the **Session** interface, but I'm going to list down a few important methods only, which we will use in this tutorial. You can check Hibernate documentation for a complete list of methods associated with **Session** and **SessionFactory**.

|  |  |
| --- | --- |
| **Sr.No.** | **Session Methods & Description** |
| 1 | **Transaction beginTransaction()**  Begin a unit of work and return the associated Transaction object. |
| 2 | **void cancelQuery()**  Cancel the execution of the current query. |
| 3 | **void clear()**  Completely clear the session. |
| 4 | **Connection close()**  End the session by releasing the JDBC connection and cleaning up. |
| 5 | **Criteria createCriteria(Class persistentClass)**  Create a new Criteria instance, for the given entity class, or a superclass of an entity class. |
| 6 | **Criteria createCriteria(String entityName)**  Create a new Criteria instance, for the given entity name. |
| 7 | **Serializable getIdentifier(Object object)**  Return the identifier value of the given entity as associated with this session. |
| 8 | **Query createFilter(Object collection, String queryString)**  Create a new instance of Query for the given collection and filter string. |
| 9 | **Query createQuery(String queryString)**  Create a new instance of Query for the given HQL query string. |
| 10 | **SQLQuery createSQLQuery(String queryString)**  Create a new instance of SQLQuery for the given SQL query string. |
| 11 | **void delete(Object object)**  Remove a persistent instance from the datastore. |
| 12 | **void delete(String entityName, Object object)**  Remove a persistent instance from the datastore. |
| 13 | **Session get(String entityName, Serializable id)**  Return the persistent instance of the given named entity with the given identifier, or null if there is no such persistent instance. |
| 14 | **SessionFactory getSessionFactory()**  Get the session factory which created this session. |
| 15 | **void refresh(Object object)**  Re-read the state of the given instance from the underlying database. |
| 16 | **Transaction getTransaction()**  Get the Transaction instance associated with this session. |
| 17 | **boolean isConnected()**  Check if the session is currently connected. |
| 18 | **boolean isDirty()**  Does this session contain any changes which must be synchronized with the database? |
| 19 | **boolean isOpen()**  Check if the session is still open. |
| 20 | **Serializable save(Object object)**  Persist the given transient instance, first assigning a generated identifier. |
| 21 | **void saveOrUpdate(Object object)**  Either save(Object) or update(Object) the given instance. |
| 22 | **void update(Object object)**  Update the persistent instance with the identifier of the given detached instance. |
| 23 | **void update(String entityName, Object object)**  Update the persistent instance with the identifier of the given detached instance. |